

# Implementación de Autenticación y Autorización con JWT y Roles en una Aplicación Web

## 1. Introducción

El presente informe detalla la implementación de un sistema robusto de autenticación y autorización basado en JSON Web Tokens (JWT) y roles de usuario para una plataforma de gestión de proyectos. El objetivo principal fue asegurar el acceso a los recursos de la aplicación únicamente a usuarios autenticados, así como controlar las funcionalidades disponibles según el tipo de usuario: Administrador, Editor y Usuario Común. La aplicación se desarrolló utilizando Node.js y Express.js para el backend, MongoDB como base de datos, y librerías clave como

jsonwebtoken para la gestión de JWT y bcrypt para el hashing de contraseñas.

## 2. Arquitectura General del Sistema

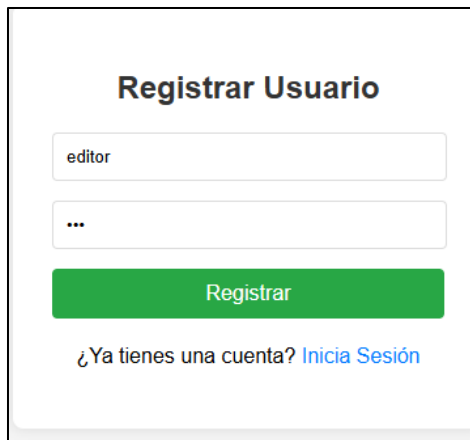
La aplicación se estructura en un modelo cliente-servidor. El frontend, compuesto por archivos HTML y JavaScript, se encarga de la interfaz de usuario y de enviar las solicitudes al backend. El backend, desarrollado con Node.js y Express.js, maneja la lógica de negocio, la interacción con la base de datos MongoDB, la autenticación, la generación y verificación de tokens JWT, y la aplicación de las reglas de autorización basadas en roles.

## 3. Implementación de Autenticación con JWT

El sistema de autenticación fue diseñado para permitir a los usuarios crear cuentas de forma segura y luego iniciar sesión para obtener un token que les confiere acceso.

- **3.1 Registro de Usuarios:** El sistema permite a nuevos usuarios crear una cuenta con un nombre de usuario y contraseña

```
// Definir el esquema y modelo de usuario
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, default: 'usuario' } // administrador, editor, usuario
});
const User = mongoose.model('User', userSchema);
```



Un formulario web con el título "Registrar Usuario" en negrita. Contiene dos campos de entrada: el primero con el texto "editor" y el segundo con tres puntos "...". Debajo de los campos hay un botón rectangular de color verde con el texto "Registrar" en blanco. En la parte inferior del formulario, hay un enlace azul que dice "¿Ya tienes una cuenta? Inicia Sesión".

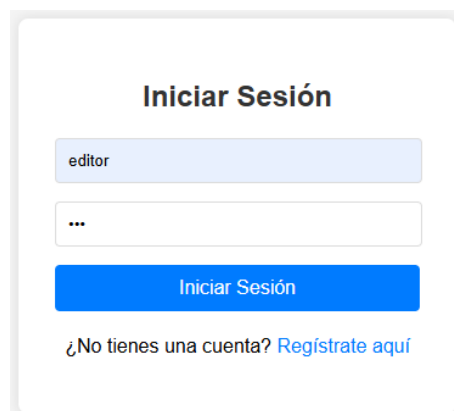
Un aspecto crítico de la seguridad es el **hashing de contraseñas** utilizando la librería bcrypt. Antes de almacenar cualquier contraseña en la base de datos, esta se somete a un proceso de hashing. Esto implica generar un "salt" (valor aleatorio) y luego combinarlo con la contraseña para producir un hash irreversible. Esto garantiza que las contraseñas nunca se almacenen en texto plano, protegiéndolas incluso si la base de datos fuera comprometida. La imagen

```
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);

const newUser = new User({
  username: username,
  password: hashedPassword,
  role: 'usuario' // Rol por defecto al registrar
});
```

Tras un registro exitoso, el sistema informa al usuario y le ofrece un enlace para ir a la página de inicio de sesión, como se ve en registroexitoso.png.

- **3.2 Inicio de Sesión y Generación de JWT:** El sistema de inicio de sesión permite a los usuarios ingresar su nombre de usuario y contraseña. Si las credenciales son correctas, el sistema genera un JWT y lo envía al cliente.



Un formulario web con el título "Iniciar Sesión" en negrita. Contiene dos campos de entrada: el primero con el texto "editor" y el segundo con tres puntos "...". Debajo de los campos hay un botón rectangular de color azul con el texto "Iniciar Sesión" en blanco. En la parte inferior del formulario, hay un enlace azul que dice "¿No tienes una cuenta? Regístrate aquí".

```
const authenticateJWT = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (authHeader) {
    const token = authHeader.split(' ')[1]; // El token viene como "Bearer TOKEN_AQUI"

    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
      if (err) {
        console.error('JWT verification error:', err);
        // Si el token es inválido o expirado, envía un 403
        return res.status(403).send('Token no válido o expirado.');
```

En el backend, se busca al usuario por su nombre de usuario y se verifica la contraseña proporcionada con la contraseña hashada almacenada utilizando bcrypt.compare. Si las contraseñas coinciden, se genera un JSON Web Token. Este JWT incluye información crucial como el ID del usuario, su nombre de usuario y, fundamentalmente, su **rol** (administrador, editor o usuario). El token tiene un tiempo de expiración limitado, en este caso, 1 hora, para mejorar la seguridad

```
PS C:\Bootcamp\2025\Laboratorio1> node .server.js  
[dotenv@17.0.0] injecting env (2) from .env - 🔒 encrypt with dotenvx: https://dotenvx.com  
Servidor escuchando en http://localhost:3000  
Ve a http://localhost:3000/register para registrarte.  
Conectado a la base de datos de MongoDB  
Nuevo token generado en servidor: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZiI6ImVudCZlcnR5IiwiaWF0IjoiYmRkMjEwMDAyZWYyOTBhbnQzOGZjc1ciInVzZCUwMTI1LjoiYWRkaWkiOiJyb2N1Ijo1YWRkaWEpc3R5IVRvc1IiImhdCI6MTUyOTcyYmVudCZlcnR5IiwidXlkIjoidjdzdmdDNDhBNHczTzF0fQ.hNvzc2LltitXXhOn6mZrH_vP2nKd8tHnVmrdhj8
```

- **3.3 Almacenamiento y Envío del JWT:** Una vez generado, el JWT se envía al cliente en la respuesta JSON del inicio de sesión. En el lado del cliente (frontend), este token se almacena en localStorage del navegador. Esta es una práctica común para Single Page Applications (SPA) y permite que el token persista a través de las recargas de página, facilitando el envío en solicitudes posteriores a rutas protegidas.

```
const user = await User.findOne({ username });

if (!user) {
  return res.status(401).send('Nombre de usuario o contraseña incorrectos.');
```

```
const isMatch = await bcrypt.compare(password, user.password);

if (isMatch) {
  const token = jwt.sign(
    { id: user._id, username: user.username, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '1h' } // EL token expira en 1 hora
  );

  console.log('Nuevo token generado en servidor:', token);
  return res.json({ message: 'Inicio de sesión exitoso', token: token, role: user.role });
} else {
  res.status(401).send('Nombre de usuario o contraseña incorrectos.');
```

## 4. Control de Acceso Basado en Roles

El sistema implementa un control de acceso granular basado en los roles de usuario definidos, asegurando que cada usuario solo acceda a las funcionalidades y datos para los que tiene permiso.

- **4.1 Roles Definidos:** Se han definido tres roles principales en la aplicación, cada uno con un conjunto específico de permisos, como se establece en las instrucciones del ejercicio:
  - **Administrador:** Puede ver, crear, editar y eliminar proyectos y tareas. Tiene acceso completo a todos los proyectos y tareas en el sistema.
  - **Editor:** Puede ver, crear y editar proyectos y tareas, pero **no puede eliminarlas**. Los editores solo pueden ver y editar los proyectos y tareas que han creado o se les han asignado.
  - **Usuario Común:** Solo puede ver los proyectos y tareas que le han sido asignados específicamente. No puede crear, editar ni eliminar.

```
_id: ObjectId('685e2806cb42f491476a8bcd')
nombre : "administrador"
usuario
contras_: "$2b$10$upb8Q/roJ15yGnilqV7rIQk8c/4SkYA7awmh3vTgH1CjGfWkp/GK"
rol : "administrador"
__v : 0

_id: ObjectId('68633ede8e2f1cd967')
nombre : "usuario"
usuario
contras_: "$2b$10$3gnn3CB3/CIMBfNDWttFYODNHytKtwc1KiPE8F5AhxN3xPqB/c3IfG"
rol : "usuario"
__v : 0

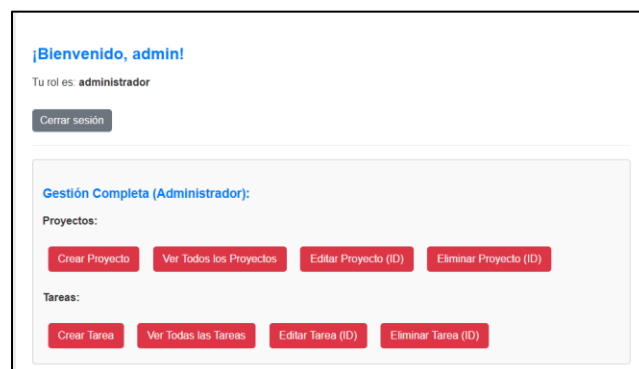
_id: ObjectId('6863738086f2b581bebfd444')
nombre : "editor"
usuario
contras_: "$2b$10$6U1CHwH16LByDj0bY71Jv0aAhgAlepx7sfwRzHf/MXkpr5OU0t5q0"
rol : "editor"
__v : 0
```

- **4.2 Middleware de Autenticación (authenticateJWT):** Este middleware es la primera línea de defensa para las rutas protegidas. Su función es interceptar cada solicitud que requiere autenticación, verificar la presencia de un JWT en el encabezado Autorización y validar su autenticidad y expiración utilizando jwt.verify. Si el token es válido, la información del usuario (incluyendo su ID, nombre de usuario y rol) se adjunta al objeto req.user, permitiendo que los siguientes middlewares o la ruta accedan a ella. Si el token es inválido o no está presente, la solicitud es rechazada con un código de estado 401 (no autorizado) o 403 (prohibido). La imagen image\_927b9a.png o image\_926c5c.png ilustra el mensaje de error cuando no se proporciona un token de autenticación.

```
// Middleware para verificar si el usuario es Administrador
const isAdmin = (req, res, next) => {
  if (req.user && req.user.role === 'administrador') {
    next(); // El usuario es administrador, permite el acceso
  } else {
    res.status(403).send('Acceso denegado. Se requieren permisos de administrador.');
```

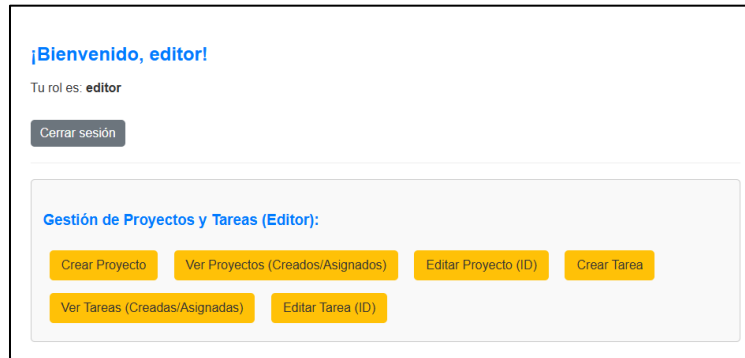
```
// Middleware para verificar si el usuario es Editor o Administrador
const isEditorOrAdmin = (req, res, next) => {
  if (req.user && (req.user.role === 'editor' || req.user.role === 'administrador')) {
    next(); // El usuario es editor o administrador, permite el acceso
  } else {
    res.status(403).send('Acceso denegado. Se requieren permisos de Editor o Administrador.');
```

- **4.3 Middlewares de Autorización (isAdmin, isEditorOrAdmin):** Estos middlewares se encargan de la lógica de autorización específica de roles. Se ejecutan después de authenticateJWT y utilizan la información del req.user.role para determinar si el usuario tiene los permisos necesarios para acceder a una ruta o recurso.
  - **isAdmin:** Permite el acceso solo si el rol del usuario es 'administrador'.
  - **isEditorOrAdmin:** Permite el acceso si el rol del usuario es 'editor' o 'administrador'.
- **4.4 Protección de Rutas y Paneles de Usuario por Roles:** Las rutas de la API están protegidas mediante la aplicación de authenticateJWT seguido de los middlewares de autorización (isAdmin, isEditorOrAdmin) donde sea necesario.
  - **Administrador:** El administrador tiene acceso a todas las operaciones (crear, ver, editar, eliminar) sobre proyectos y tareas. Las rutas sensibles como la eliminación de proyectos o tareas están protegidas con authenticateJWT y isAdmin. La imagen login-admin.png muestra el dashboard actualizado para un usuario administrador, presentando los botones de "Gestión Completa" para proyectos y tareas. Anteriormente, el panel-admin.png mostraba una interfaz similar pero con "Gestión de Publicaciones".

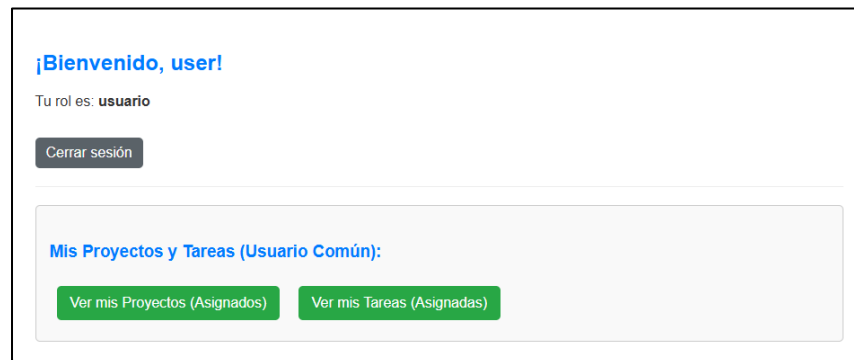


Para rutas de eliminación, se usaría isAdmin en lugar de isEditorOrAdmin.

- **Editor:** El editor puede crear, ver y editar proyectos y tareas, pero no tiene permisos para eliminarlas. Las rutas correspondientes están protegidas por authenticateJWT y isEditorOrAdmin.



- **Usuario Común:** El usuario común solo tiene permiso para ver los proyectos y tareas que le han sido asignados. No puede realizar operaciones de creación, edición o eliminación. La interfaz del dashboard se adapta para mostrar únicamente estas opciones de visualización. La imagen



La lógica para filtrar los proyectos y tareas mostrados a usuarios comunes y editores se implementa directamente en las rutas GET /api/projects y GET /api/tasks, consultando la base de datos de manera condicional basada en req.user.role.

```
// Crear un nuevo proyecto (solo Administrador o Editor)
app.post('/api/projects', authenticateJWT, isEditorOrAdmin, async (req, res) => {
  const { name, description, assignedUsers } = req.body;
  try {
    const newProject = new Project({
      name,
      description,
      createdBy: req.user.id, // El ID del usuario que crea el proyecto (del JWT)
      assignedUsers: assignedUsers || [] // Permite asignar usuarios al crear
    });
    await newProject.save();
    res.status(201).json(newProject);
  } catch (error) {
    console.error('Error al crear el proyecto:', error);
    res.status(500).send('Error en el servidor al crear el proyecto.');
```

## 5. Gestión de Sesiones (Cierre de Sesión)

La funcionalidad de cierre de sesión es crucial para la seguridad, permitiendo al usuario invalidar su sesión activa en el cliente.

- **Cierre de Sesión:** La implementación del cierre de sesión es sencilla en un sistema basado en JWT. Consiste en eliminar el JWT almacenado en el localStorage del cliente. Una vez que el token es eliminado, cualquier intento posterior de acceder a rutas protegidas resultará en una denegación de acceso (ya que no se enviará un token o el token estará ausente/inválido). Esto efectivamente "cierra la sesión" del usuario en ese navegador.

```
// Cierre de sesión (simplemente elimina el token y redirige)
document.getElementById('logoutLink').addEventListener('click', (event) => {
  event.preventDefault();
  localStorage.removeItem('jwtToken');
  alert('Has cerrado sesión.');
```

```
window.location.href = '/login';
});
```

## 6. Conclusiones

El ejercicio ha permitido implementar satisfactoriamente un sistema completo de autenticación y autorización utilizando JWT y roles de usuario. Se lograron los siguientes objetivos:

- Un sistema de autenticación seguro que permite el registro y el inicio de sesión, con contraseñas protegidas mediante bcrypt.
- La generación y validación de JWT para la gestión de sesiones, incluyendo el rol del usuario y una expiración.
- Un robusto control de acceso basado en roles (administrador, editor, usuario), que restringe las funcionalidades y la visibilidad de los datos según los permisos del usuario.
- La protección efectiva de rutas en el backend a través de middlewares de autenticación y autorización.
- La implementación de una función de cierre de sesión que invalida la sesión del lado del cliente.

Este proyecto demuestra una comprensión sólida de los principios de seguridad en aplicaciones web modernas, incluyendo la gestión de sesiones sin estado (gracias a JWT) y la implementación de un control de acceso basado en roles, fundamental para aplicaciones con diferentes niveles de privilegios.